MOLTAP : A Modal Logic Tableau Prover
Users guide

Twan van Laarhoven

10th April 2008

1 Web interface

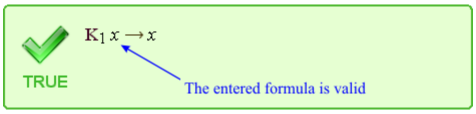
1.1 Entering a formula

By default the prover web interface shows a single line textbox where a formula can be entered. See the syntax reference for a description of the syntax to use. Press `Enter` to evaluate the formula.

When a single line does not suffice, the input field can be expanded by clicking the downward pointing arrow on the right.

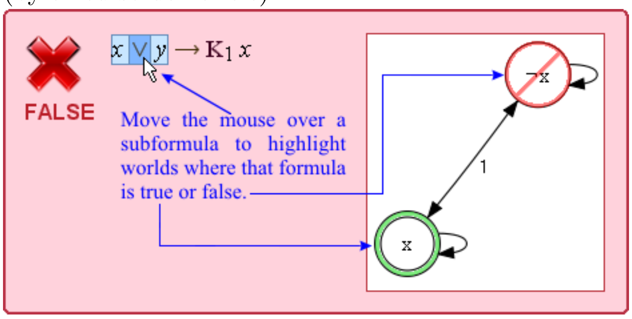
1.2 True formulas

When a formula is valid in all worlds MOLTAP will show a green box.



1.3 False formulas

For formulas that are not valid in all worlds MOLTAP will show a red box with a counter model. In this model the formula is false. When the mouse is moved over a subformula the program will indicate in which worlds this subformula is true (by a green circle) and in which worlds it is false (by a red struck circle).



2 Command line program

MOLTAP also comes with a command line version. The syntax is exactly the same as for the web interface. This program supports three modes of operation

1. Read input from a file or stdin.
2. Read input from the command line.
3. A simple interactive mode.

In each case one or more formulas are evaluated (proven/disproven). The program writes `true` or `false` to the output, and if the formula is false generates a counter model. The command line program supports the following arguments

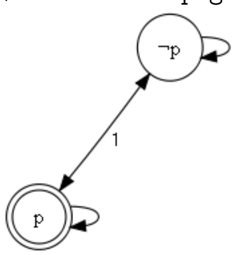
Short form	Long form	Description
`FILE`		Run the program on the given input file.
`-?`	`--help`	Show help page.
`-i`	`--interactive`	Interactive mode.
`-f FORMULA`	`--formula=FORMULA`	Give a formula directly on the command line.
`-o FILE`	`--model-name=FILE`	Filename for generated model images, the default is “model.png”. The extension determines the generated image type.

When reading input from the command line the end of file character must be used to indicate the end of the formula, use `^Z` on windows and `^D` on linux.

In interactive mode each line is considered to be a formula, unless there are remaining parentheses to be closed. Use `:?` to show the help page and `:q` to quit.

2.1 Example session

```
$ moltap
x | x
^Z
true
$ moltap -f "p -> K1 p"
false
$ view model.png
```




```
$ moltap -i
> K1,2 p -> (
..  K1 p & K2 p
..  )
true
> :q
Goodbye
```

3 Syntax reference

3.1 Propositional formulas

There are three ways to write logical connectors

1. Using ASCII syntax, for example `p & q`.
2. Using Unicode symbols, for example `p ∧ q`.
3. Using natural language, for example `p and q`.

Input syntax			Description
ASCII	Unicode	Text	
`p`, `q`, `cat`, `bigVar123`			Propositional variables consists of alphanumeric symbols, starting with a lower case letter.
`(`φ`)`			Parentheses can be used for grouping.
	⊤, ⊥	`true`, `false`	The true/false proposition.
`~`φ	¬φ	`not` φ	Logical negation, $\neg\varphi$ is true if and only if φ is false.
φ `&` ψ	φ ∧ ψ	φ `and` ψ	Logical conjunction
φ `|` ψ	φ ∨ ψ	φ `or` ψ	Logical disjunction
φ `->` ψ	φ → ψ	φ `implies` ψ	Logical implication
φ `<-` ψ	φ ← ψ		Implication written the other way around.
φ `<->` ψ, φ `=` ψ	φ ↔ ψ		Logical equivalence
φ `<-/->` ψ, φ `/=` ψ	φ ↮ ψ, φ ≠ ψ		Logical inequivalence

3.2 Modal formulas

Modal formulas are formulas about agents. Agent names can be arbitrary strings of alphanumeric symbols. Examples of valid agent names are

- `1`, `2`
- `Alice`, `Bob`
- `my_Computer`
- α, β

MOLTAP supports both the epistemic style (`K`/`M`) and modal style (□/◇) of writing operators.

Input syntax		Description
Epistemic	Modal	
`K_1` φ, `K1` φ	`[]1` φ, `□1` φ	Agent 1 knows that φ, φ is necessary for agent 1.
`K{1,2}` φ	`[]{1,2}` φ	Agents 1 and 2 both knows that φ. This is the same as `K1` φ `&` `K2` φ.
`K` φ	`[]` φ	Every agent knows φ.
`K*` φ, `K*{1,2}` φ	`[]*` φ	It is common knowledge that φ. Every agent (in a set) knows that φ, and they know that everyone knows, and they know that everyone knows that everyone knows, etc.
`M_1` φ, `M1` φ	`<>1` φ, `◇1` φ	Agent 1 holds φ for possible, φ is possible for agent 1.

3.3 Advanced features

Input syntax	Description
`let x =` φ`;` ψ	Create a local declaration. Inside ψ all occurrences of `x` will be replaced by φ. Declarations are only allowed at the top level.
`system S;` ψ	Change the axiom system. By default system S5 (=KT45) is used. The name S is either a combination of KDT45 or S4 or S5.
`# comment`	Comments begin with a `#` sign and run until the end of the line.

The possible axioms stand for:

- K = the basic axiom system, this is always used.
- D = seriality, each world has a successor, $\Diamond_i\top$.
- T = all relations are reflexive, $\Box_i\varphi \rightarrow \varphi$.
- 4 = all relations are transitive, $\Box_i\varphi \rightarrow \Box_i\Box_i\varphi$.
- 5 = all relations are euclidean, $\Diamond_i\varphi \rightarrow \Box_i\Diamond_i\varphi$.

3.4 Precedence and associativity

1. `not` and modal operators bind the strongest.
2. Followed by `and`,
3. then `or`
4. and finally implication and equivalence. Implication associates to the right.
5. Programming features like `let` bind even weaker.

So for example `K1 p ∧ ~q ∨ r → s → t` is parsed as `(((K_1 p) ∧ (¬q)) ∨ r) → (s → t)`.